Multi-objective engineering shape optimization using differential evolution interfaced to the Nimrod/O tool



2010 IOP Conf. Ser.: Mater. Sci. Eng. 10 012189

(http://iopscience.iop.org/1757-899X/10/1/012189)

IOP Conf. Series: Materials Science and Engineering **10** (2010) 012189 doi:10.1088/1757-899X/10/1/012189

# Multi-objective engineering shape optimization using differential evolution interfaced to the Nimrod/O tool

**Mike J W Riley[1], Tom Peachey[2], David Abramson[2] and Karl W Jenkins[1]**

[1]Applied Mathematics and Computing Department, Cranfield University, BEDS, MK43 0AL, United Kingdom

[2]Faculty of Information Technology, Monash University, CLAYTON, VIC 3800, Australia

Email: m.riley@cranfield.ac.uk, tom.peachey@infotech.monash.edu.au, david.abramson@infotech.monash.edu.au, k.w.jenkins@cranfield.ac.uk

**Abstract**. This paper presents an enhancement of the Nimrod/O optimization tool by interfacing DEMO, an external multiobjective optimization algorithm. DEMO is a variant of differential evolution – an algorithm that has attained much popularity in the research community, and this work represents the first time that true multiobjective optimizations have been performed with Nimrod/O. A modification to the DEMO code enables multiple objectives to be evaluated concurrently. With Nimrod/O's support for parallelism, this can reduce the wall-clock time significantly for compute intensive objective function evaluations. We describe the usage and implementation of the interface and present two optimizations. The first is two-objective mathematical function in which the Pareto front is successfully found after only 30 generations. The second test case is the three-objective shape optimization of a rib-reinforced wall bracket using the Finite Element software, Code_Aster. The interfacing of the already successful packages of Nimrod/O and DEMO yields a solution that we believe can benefit a wide community, both industrial and academic.

## 1. Introduction

Optimization continues to be a widely researched topic and finds applications throughout science and engineering disciplines in both academia and industry[1]. It is often the case that two or more objectives need to be optimized simultaneously. A true multiobjective optimization process will not produce one single solution if any of the objectives are in conflict with each other. Mathematical optimization techniques have existed since the 18th century when Leibniz and Euler used differential calculus to develop a tool for evaluating minima and maxima of differentiable relationships, however it was not until the French-Italian economist V. Pareto (1848-1923) developed his theory of Pareto optimality that a framework could exist for multiobjective optimization problems (MOOP)[2]. The defining characteristic of the Pareto optimal set is the loss of optimality in one objective function as another objective function is improved.

Since the 1980's, sufficient computing power has existed to approach the MOOP via the use of bio-inspired metaheuristics. The focus of optimization has shifted from mathematical programming techniques to the application of evolutionary methods, which adapt the genes of a population of

 

candidates with the aim of improving their "fitness". Mathematical programming techniques typically require multiple optimization runs, each generating one element of the Pareto set and are susceptible to changes in the shape of the Pareto front (Section 2.2) and may not work when this front is concave or discontinuous[1]. By contrast, evolutionary algorithms simultaneously manipulate a set of possible solutions. In addition, evolutionary algorithms typically deal with discontinuous or concave Pareto fronts much better[1]. For this reason, they are known as "robust" optimization methods. Examples include: Strength Pareto Evolutionary Algorithm (SPEA)[3], Non-dominated Sorting Genetic Algorithm (NSGA)[4], Multi-Objective Tabu Search (MOTS)[5], and Differential Evolution Multi-Objective (DEMO)[6].

However, executing an optimization involves much more than just applying a suitable algorithm. This is especially true when the calculation of the objective function is very (time) expensive - for example when the objective function is the result of a complex numerical simulation. Cluster and grid/cloud computing provide institutions with access to more much computing power than any one workstation, but all the elements of the optimization loop must somehow be coordinated and managed. In addition, the parameters of an optimization, and any constraint conditions, have to be described to the optimization software before it can be initiated. Part of a suite of problem solving tools developed at Monash University, Nimrod/O [7] is the optimization element and can be interfaced with computing clusters or grids [8] for the parallel evaluation of expensive objective functions. Using its declarative "plan file", a user can specify an optimization readily. Nimrod/O has long provided a range of single objective optimization algorithms, and multiobjective optimizations have been possible by re-phrasing them as single objective optimizations with the use of a cost function. This paper represents the first interface of a true multiobjective optimization algorithm.

The layout of the paper is as follows: The Nimrod/O optimization tool and the chosen optimization algorithm, DEMO, are described in more detail in sections 2.1 and 2.2. Adaptations to DEMO that enable parallelism, and the role of the DEMOinterface are detailed in section 2.3 before the two test experiments are presented (3.1 and 3.2). The first test in section 4.1 is a two-parameter optimization of a mathematical test function. The second test in section 4.2 is the shape optimization of a rib-reinforced steel bracket using Finite Element evaluations from Code_Aster to compute the two objective functions of stress and deflection as well as incorporating a third, conflicting objective function, of reducing the mass of the part.

## 2. Software components

### *2.1. Nimrod/O*

Nimrod/O combines optimization, distributed computing and rapid prototyping in one tool. Various optimization routines are built into Nimrod/O such as BFGS (Broyden–Fletcher–Goldfarb–Shanno), the Downhill Simplex Method, Simulated Annealing, and EPSOC (Evolutionary Programming using Self-Organised Criticality)[9]. An optimization is readily specified by the user by parameterizing their problem using Nimrod/O's declarative "plan file" (Figure 3), after which the tool computes the parameters that minimize or maximize the design's objective function. Transparent to any of the optimization algorithms is Nimrod/O's evaluation of the objective function. These evaluations can execute in parallel; on a multi-core CPU on the local machine, be farmed out to greater compute resources such as a cluster (e.g. [10]), or a grid resource such as provided by Nimrod/G[8], as shown in Figure 1.

Nimrod/O has been applied to a wide range of problems since its publication **[**11**]**, such as aerofoil optimization, air pollution studies, stress analysis, cardiac modelling, structural geology, fatigue based design optimization and high gain antenna design [12]**.** An additional flexibility of the tool is the provision for the user to incorporate their own optimization algorithm. These algorithms can be specified as a user-defined function, in which communication is facilitated by passing memory structures, or, implemented as a program external to Nimrod/O when pipes are used for the cross process communication. Recent code revisions of Nimrod/O have enabled multiobjective

optimizations to be handled and the first interface of a robust multiobjective optimization package is the subject of this paper.

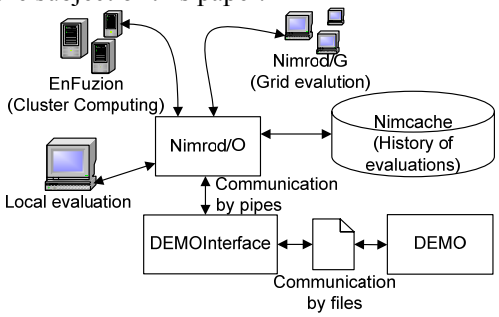


**Figure 1.** Overview of the process.

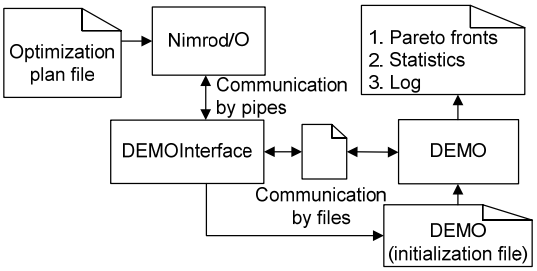


**Figure 2.** Dataflow between the software elements.

*2.2. DEMO*

Differential evolution (DE) by Price [13] was the culmination of work aimed at solving the Tchebychev Polynomial fitting problem proposed to him by Dr R Storn. It is a population-based optimization algorithm, but unlike classical genetic algorithms such as Holland's [14] which encodes decision variables as binary numbers, DE uses real coding of floating point numbers. This, coupled with Price's desire to make candidate mutation an adaptive procedure, resulted in a rapid and robust algorithm that is simple to use. The original version of DE is controlled by just three variables: the population size, N, the mutation scaling Factor, F, and the crossover constant, CR.

Many optimization problems involve more than one criterion. In the unlikely event that these objectives are *not* in conflict with each other, the multiobjective optimization problem can reduce to a single ideal solution. However, this is not normally the case – for example the mass of an engineering part is desired to be low, whilst its load carry capabilities are desired to be high. In this situation, no single optimal solution will exist and the optimization algorithm searches for the *Pareto front*. The minimization of a general two criteria multiobjective optimization is formulated as follows:

Minimize $f(x) = (f_1(x), f_2(x))$ such that $x \in X$, the feasible region

subject to $\begin{cases} g_j(x) = 0 \;\; j = 1, \dots, M \\ h_k(x) \leq 0 \;\; k = 1, \dots, K \end{cases}$ constraints

where x is a p-dimensional vector whose components are know as decision variables, $g_j$ are equality constraints and $h_k$ are inequality constraints.

Definition of *dominance*: Comparing two solutions, $x_1$ and $x_2$, we say that $x_1$ dominates $x_2$ if:

$$f_1(x_1) < f_1(x_2) \;\text{ and }\; f_2(x_1) \leq f_2(x_2) \qquad \text{or} \qquad f_1(x_1) \leq f_1(x_2) \;\text{ and }\; f_2(x_1) < f_2(x_2)$$

The *Pareto set* is formed from only those solutions that are not dominated by any other (i.e. from *non-dominated solutions).* The *Pareto front* is an imaginary line drawn in the objective space, along which candidates from the *Pareto set* would lie.

Consequently, two goals exist in multiobjective optimization[6]:

1. Find the most diverse range of these solutions across the Pareto set, and
2. Discover solutions as close as possible to the ideal Pareto front.

To these ends, Robič and Filipič developed DEMO (Differential Evolution for Multiobjective Optimization)[6]. Based on DE, it builds on the success of Price's algorithm and adds the mechanisms of non-dominated sorting and crowding distance metric as used by state-of-the-art multiobjective evolutionary algorithms. This helps achieve the first goal of finding the most diverse range of non-dominated solutions. The second goal is achieved by an emphasis on elitism: parent individuals are immediately replaced by the candidate that dominates them. By entering the population immediately, this new candidate can, without delay, take part in the creation of further candidates. With these additions, DEMO is shown to achieve competitive results on five ZDT [15] test problems. In a follow up paper, Robic[16] presents a comparison study in which DEMOS's performance is found to be comparable to other state-of-the-art multiobjective evolutionary algorithms on nine newer test problems created by Huband et al. [17].

### *2.3. Interfacing DEMO with Nimrod/O*

The original DEMO code was first ported from its MS Windows source code such that it could compile under the Linux operating system. The random number generator, a container declaration and the system-out calls comprised the three necessary alterations. Initial testing confirmed that the Linux port of DEMO worked equivalently to the Windows version.

As described in section 2.2, one of DEMO's key mechanisms is elitism within the reproduction process. Before an entire population has been evaluated, superior candidates will already have replaced their parents and take part in the creation of newer candidates. It should be clear that this mechanism requires sequential candidate evaluation and presents a conflict of interest. Whilst this elitism mechanism accelerates the discovery of the Pareto optimal set, parallel candidate evaluations would reduce the wall-clock time for optimization runs. To this end we introduce to DEMO's initialization file a "*BatchSize*" parameter. $\text{BatchSize} \leq P \leq N$ where, $N =$ Population size, and $P =$ Number of processors available for parallel objective function evaluations.

In the case that the user has access to a large compute resource, the BatchSize parameter tunes-down the benefit from elitism in favor of an overall speed-up from parallelism. One further minor change to DEMO's initialization file is the inclusion of a Boolean flag that indicates to DEMO that it will be working in a mode compatible with Nimrod/O. If this flag is turned off (0), then DEMO will function in stand-alone mode and identical to version 1.2. More information on DEMO's usage can be found in the v.1.2 reference manual [18].

Nimrod/O 2.9 now supports multiobjective function evaluations. Via a "`results`" parameter in the plan file (Figure 3), Nimrod/O prepares to accept multiple objective functions and, during run time, both logs and caches these multiple results. As in prior versions, the cache mechanism (Figure 1) prevents unnecessary repetitions of prior function evaluations. The management of Pareto optimal sets, Pareto based ranking and sorting is not supported by the current version of Nimrod/O, however DEMO provides this functionality.

Nimrod/O can host a concurrent execution thread in which an external optimizer runs. This intent is communicated in the plan file by the use of "`method external "name" ./executable`". For this paper, the pipes method was chosen. In building the interface, the necessary *include files* from Nimrod/O's package were `noclient.c`, `noclient.h` and `definitions.h`. These provide query and communication functionality between external code, such as the current interface, and Nimrod/O. Sufficient functions are implemented in `noclient.c` that an external, user-defined, optimization algorithm can operate as if it were part of Nimrod/O.

The DEMOinterface is simultaneously the child process of Nimrod/O and the parent process of DEMO and, in use; it translates data formats and requests between these two applications (Figure 2). The user may alter specifics of the DEMO optimization by editing DEMO's initialization file. For the convenience of the user, fields in Nimrod/O's plan file that are repeated in DEMO's initialization file are automatically inserted into DEMO's initialization file by the interface before it spawns DEMO.

The *stopping criterion* for DEMO is specified in its initialization file as a maximum number of candidate evaluations. Once this limit is reached, DEMO writes the current Pareto front to a file called

`fronts.out`. Further files such as the statistics on the population's evolution and a log file are written by DEMO before it terminates. The DEMOinterface also detects when the maximum number of evaluations has been reached and notifies Nimrod/O which likewise finalizes its files and terminates.

In addition to creating the DEMOinterface, the authors of this paper have made the above alterations to DEMO and further developed Nimrod/O for multiobjective compatibility. The rest of this paper concerns; testing the solution by minimizing a two-objective mathematical function, and, the three objective shape optimization of an engineering part using the Finite Element package, Code_Aster.

## 3. Experimental setup

### *3.1. Poloni test function*

Poloni's function [19] offers a convenient way to test the DEMO algorithm. It is a two parameter, two response, mathematical function (given below).

$$x, y \in [-\pi, \pi]$$

$$F_1(x,y) = -[1 + (A_1 - B_1)^2 + (A_2 - B_2)^2]$$

$$F_2(x,y) = -[(x+3)^2 + (y+1)^2]$$

Where,

$$A_1 = 0.5\sin 1 - 2.0\cos 1 + \sin 2 - 1.5\cos 2$$

$$A_2 = 1.5\sin 1 - \cos 1 + 2\sin 2 - 0.5\cos 2$$

$$B_1 = 0.5\sin x - 2\cos x + \sin y - 1.5\cos y$$

$$B_2 = 1.5\sin x - \cos x + 2\sin y - 0.5\cos y$$

Price [13] provides a guide for the population size as $10 \times d$ where d is the number of dimensions of the problem, therefore in this test $N = 20$ . The weight of the mutation scaling factor can be any value in the interval [0,2] and was chosen as $F = 0.5$. The crossover probability must lie in the interval [0,1] and $CR = 0.3$ was chosen. These F and CR values were used for both the optimizations presented in this paper. Price and Storn [20] describe the settings for these parameters in more detail. A concurrency setting of 4 directed Nimrod/O to enable parallel function evaluations on the quad-core host machine.

```
parameter p float range from -3.1415926 to 3.1415926
parameter q float range from -3.1415926 to 3.1415926

results 2

task main
   copy  poloni node:poloni
   node:execute  ./poloni $p $q
   copy  node:exp_result output.$jobname
endtask

method external "DEMO" ./DEMOinterface
  starts 1
  endstarts
endmethod
```

**Figure 3.** Nimrod/O plan file: poloni.shd.

### *3.2. Shape optimization of a rib-reinforced wall bracket*

The shape under consideration is a rib-reinforced wall bracket. The back face of the bracket is constrained and a distributed loading is applied to the protruding face, simulating the bracket supporting a weight of approximately 200kg. Technical drawings (Figures 5 and 6) show the dimensions of the part (mm) as well as the five decision variables $(A, \ldots, E)$. These variables will be optimized to minimize the three objective functions of: mass, maximum deflection, and, maximum VonMises stress. Minimizing the mass conflicts with minimizing the stress and the deflection and so the problem will not reduce to one optimal solution – instead a Pareto set of solutions will be found.

**Table I.** Wall bracket decision variables.

$A$ = Thickness of bracket plate (mm) [1,10]
$B$ = Thickness of ribs (mm) [1,10]

$C$ = Placement of ribs (%). When:
$C = 0$, Rib distribution is widest
$C = 100$, Rib's Inner faces are 10mm from mounting holes

$$Absolute\_Offset\_OuterFace_{Rib\_1} = 119 - \left(\frac{C}{100} \times (29 - B)\right) \quad (1)$$

$$Absolute\_Offset\_OuterFace_{Rib\_2} = 1 + \left(\frac{C}{100} \times (29 - B)\right) \quad (2)$$

$D = x$ displacement of curve control point [30,70]
$E = y$ displacement of curve control point [30,70]

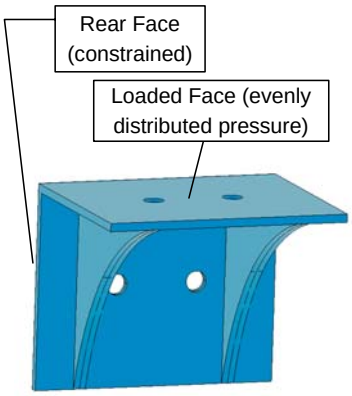


**Figure 4.** Rib-reinforced wall bracket

A stand alone computer was used for the results in this paper with a Quad Core AMD Phenom 2.5GHz processor, 4MB cache with 4GB of RAM installed. The operating system was CAELinux2008 [21] which includes the open-source CAE software: Salomé, Code_Aster, Code_Saturne and OpenFOAM. For this work, only Salomé and the Finite Element software of Code_Aster were used. Onto the base installation of the operating system, the source codes for Nimrod/O 2.9, DEMOinterface and DEMO were compiled and installed.

The five decision variables in Table I comprise the thickness of the bracket, A, and the thickness of the ribs, B, a distribution of the ribs, C, and co-ordinates for a curve control point D and E. The distribution of the ribs is presented to the optimizer as a floating point variable in the range [0,100], however this variable needs to be translated into physical dimensions on the bracket itself. The equations used to translate the variable C are equations (1) and (2)**.** These equations are necessary to accommodate changes to the rib thickness, B, and guarantee that when C=100 the inner faces of both ribs will be exactly 10mm from the centre of the mounting holes irrespective of the value B (Figure 5). Likewise, when C=0, the outer faces of the ribs will be located at their widest distribution: 1mm from the outer edges of the bracket itself. Both ribs are symmetrically distributed. D and E are $x$ and $y$ co-ordinates of a point through which the profile of the ribs is interpolated. D and E are in the interval [30, 70], the 30 being the displacement in mm from the inner face of the bracket therefore keeping the rib profile point independent of A (Figure 6)**.**

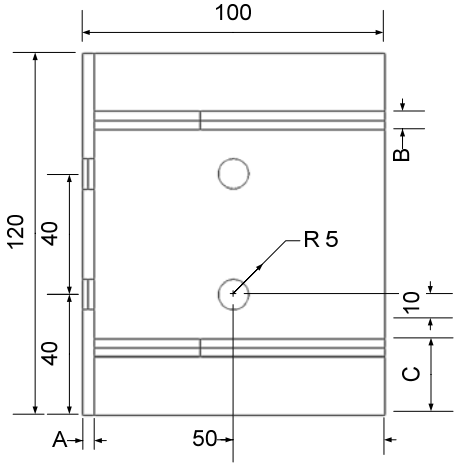


**Figure 5.** Plan view of the wall bracket.

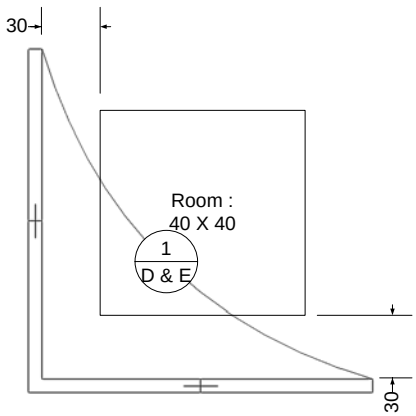


**Figure 6.** Side elevation of the wall bracket.

3.2.1. *The shape optimization job.* The flow chart, Figure 7 shows the steps involved for shape optimization using Code_Aster, Nimrod/O and DEMO. The first two steps involved setting up the shape and the optimization, but the main work was conducted in an automated loop governed by Python scripts and simple shell scripting.

From the Graphical User Interface (GUI) of Salomé, arbitrary settings for the decision variables $(A, \ldots, E)$ were chosen in building the body of the first shape. The geometry was auto meshed with the in-built algorithms shown in Table III. The volume contained ~ 11,000 Tetrahedrons after meshing. The Code_Aster Linear Elastic job was set up with a distributed pressure loading of 0.16667 MPa that represents ~200kg mass on to the upper surface of the bracket. The degrees of freedom for the rear face and interior of the rear bolt holes is given by (DX,DY,DZ) = (0,0,0). The relevant physical properties of the chosen material, Plain Carbon Steel, are given in Table II. After verifying a successful run of the Code_Aster solver, the above three steps were "dumped as Python study". In this way templates were created that could later be called from the command line.

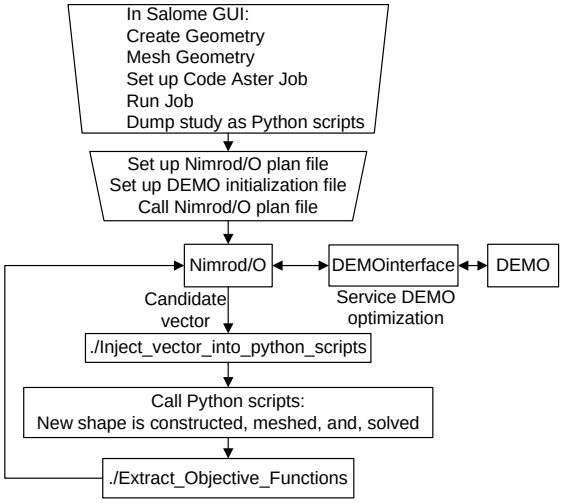


**Figure 7.** Flowchart of the shape optimization process.

**Table II.** Material properties of the bracket.

| Plain Carbon Steel | |
|---|---|
| Young's modulus, E $(GPa)$ | 200 |
| Poisson's ratio, $\upsilon$ | 0.3 |
| Density $(g/cm^3)$ | 7.86 |
| Yield Stress, $\sigma_y$ $(MPa)$ | 280 |

**Table III.** Auto meshing settings used.

| Meshing | Applied Algorithms | Applied hypotheses |
|---|---|---|
| 1D | Average length (6) | Wire discretisation Added: Quadratic Mesh |
| 2D | MEFISTO_2D | Length from edges |
| 3D | Tetrahedron (Netgen) | |

Two edits were then necessary in the text files *name*`.comm` and *name*`GEMO.py`. In the *name*`.comm` text file, maximum deflections and principal stresses were requested to be included in the pain text *name*`.resu` results file of Code_Aster. In the Python geometry script, *name*`GEMO.py`, the following lines were added adjacent to the last line:

```
myTuple = geompy.BasicProperties(finished_body)
myMass = (myTuple[2]/1000) x 7.86
```

This calculates the volume of the shape and multiplies by the density. Further Python commands save this mass to file. Two simple C++ programs were also written. `Inject_vector_into_python_scripts` takes the decision variables $(A, \ldots, E)$ as arguments, parses the template of *name*`GEMO.py`, and inserts changes to the geometry script at runtime. `Extract_Objective_Functions` is called after Code_Aster, extracting the calculated values for maximum deflection and the maximum VonMises stress from *name*`.resu`. The mass is also read-in from file, and the three objective functions are then formatted for Nimrod/O by `Extract_Objective_Functions` and saved to file. After setting up Nimrod/O's plan file and DEMO's initialization file, a small number of shell scripts were created to implement automation. The

memory requirement for an individual job was ~1.3GB. With the installed 4GB of RAM, and with the operating system overhead, a concurrency setting of 2 was the maximum level of parallelism attainable without paging to the hard disk. 6GB or more of RAM would have permitted four concurrent shape evaluations.

## 4. Results

### *4.1. Poloni test function*

Figure 8 shows a scatter plot of the Poloni function. 600 function evaluations were performed by Nimrod/O and the final Pareto set of 20 candidates found by DEMO is shown with superimposed square diamond markers. An interpolated line has been added to aid clarity. Visual inspection of this Pareto set indicates that DEMO has been successful in attaining the two aims of; finding a diverse range of solutions, and, finding solutions that are as close as possible to the ideal Pareto front. Arguably, this front is superior to that obtained by Poloni et al. [19] with their MOGA (Multi Objective Genetic Algorithm) optimizer which involved 50 candidates and 2500 evaluations.

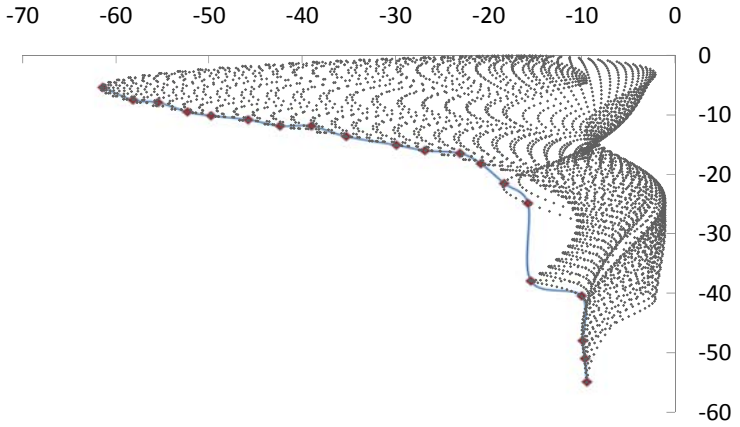


**Figure 8.** Poloni function, Pareto set superimposed.

### *4.2. Shape optimization of the rib-reinforced wall bracket*

800 candidate evaluations were performed by Nimrod/O, each involving the creation of new geometries and a linear elastic simulation by Code_Aster. The population size was N=50 and four results from the final Pareto set are given in Table IV. Across the final, 50 candidate Pareto set, the decision variables fell in the intervals: A[1.0,10.0] B[1.0,5.58] C[86.6,97.7] D[30.0,67.0] E[30.0,57.1]

The full Pareto set is plotted in the 3D scatter graph, Figure 9, showing mass, maximum VonMises stress and maximum deflection on each axis. In Table IV, displayed are the two heaviest candidates among the Pareto set for which calculations of maximum VonMises stress and maximum displacement were least. The lightest candidate was found to have a maximum VonMises stress of only 3% below the $\sigma_y$ of 280MPa. A typical safety factor setting of 3.0 would exclude this bracket from use, and likewise the next 16 light-weight solutions due to high imposed stresses. By inspection of the scatter graph in Figure 9, there is a region containing a small number of candidates (lying near the point where the mass begins to significantly increase) that substantially reduce the stress and deflection when compared to the lightest candidates. One of these is labeled the "compromise solution" (Table IV). For this candidate, the maximum calculated VonMises stress is 15.2% of the $\sigma_y$ and the Mass is only 28.3% of the two heaviest solutions. The deflections of this compromise solution are represented visually in Figure 10**.** The greatest deflections of this solution are located in the 50% of the loaded face that is furthest from the back plate, at the extreme left and right edges.

**Table IV.** Results of the multiobjective wall bracket optimization.

| Decision variables | | | | | Objective functions | | | |
|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | Max Deflection (μm) | Max VonMises (MPa) | Mass (kg) | |
| 1.00 | 1.00 | 91.1 | 30.0 | 33.0 | 739 | 271 | 0.22 | Least Mass |
| 10.0 | 4.94 | 97.7 | 40.1 | 30.0 | 0.90 | 3.82 | 1.99 | Least Stress |
| 10.0 | 5.44 | 91.7 | 33.8 | 30.0 | 0.81 | 3.90 | 1.98 | Least Deflection |
| 2.71 | 1.00 | 90.4 | 40.1 | 41.4 | 29.4 | 42.5 | 0.56 | Compromise solution |

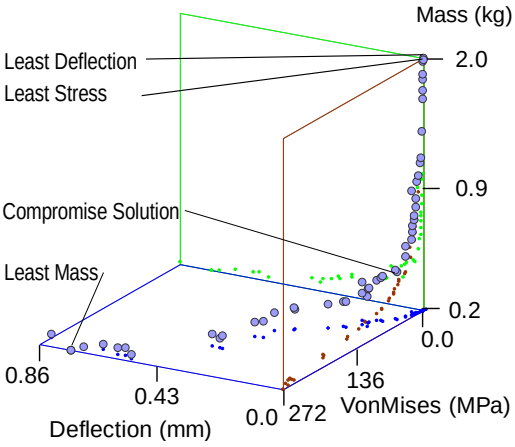


**Figure 9.** 3D scatter plot of the Pareto optimal set.

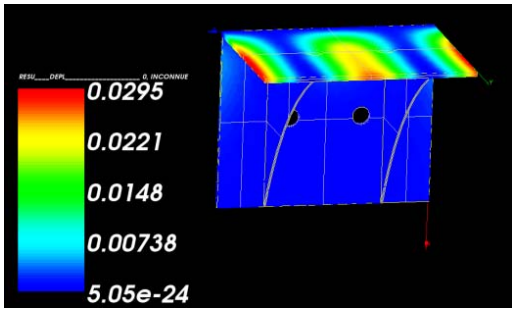


**Figure 10.** Deflections of the compromise solution (Key in mm).

## 5. Conclusion

This paper has described the successful implementation of the DEMO - Nimrod/O interface and illustrated its usage with two, truly multiobjective, optimizations. A parameter that enables parallelism has been implemented which can reduce the wall-clock time for optimizations when multiple processors are available, or, be tuned-out by the user - potentially accelerating the convergence to the Pareto front.

## 6. Acknowledgments

This work was supported by an EPSRC doctoral training grant.
The authors are grateful to Bogdan Filipič and Tea Tušar for creating DEMO and sharing their code.

## References

[1] Coello C A 2006 Twenty years of evolutionary multi-objective optimization: A historical view of the field *IEEE Computational Intelligence Magazine* **1** 28–36
[2] Maalawi K Y and Badr M A 2009 Design Optimization of Mechanical Elements and Structures: a Review with Application *Journal of Applied Sciences Research* **5(2)** 221–231
[3] Zitzler E and Thiele L 1993 Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach *IEEE Transactions on Evolutionary Computation* **4** 292–301
[4] Srinivas N and Deb K 1994 Multiobjective Optimization Using Nondominated Sorting in

Genetic Algorithms *Evolutionary Computation* **2(3)** 221–248
[5] Jaeggi D M Parks G T Kipouros T and Clarkson P 2008 The development of a multi-objective tabu search algorithm for continuous optimisation problems *in EJOR feature issue on Adaptation of Discrete Metaheuristics for Continuous Optimization* **185** 1192–1212
[6] Robic T and Filipic B 2005 DEMO: Differential Evolution for Multiobjective Optimization *Third International Conference on Evolutionary Multi-Criterion Optimization* **3410** 520–533
[7] Abramson D Lewis A Peachey T and Fletcher C 2001 An Automatic Design Optimization Tool and its Application to Computational Fluid Dynamics *in Proceedings of the Super Computing 2001 Conference*
[8] Abramson D Giddy J and Kotler L 2000 High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? *International Parallel and Distributed Processing Symposium* 520–528.
[9] Peachey T 2005 *Nimrod/O User Manual* http://messagelab.monash.edu.au/NimrodO/Documentation
[10] Axceleon 2010 *EnFuzion* http://www.axceleon.com/
[11] Abramson D Lewis A and Peachey T 2000 Nimrod/O: A tool for Automatic Design Optimization *in Proceedings of The 4th International Conference on Algorithms & Architectures for Parallel Processing*
[12] Monash eScience Applications *http://messagelab.monash.edu.au/EScienceApplications*
[13] Price K V 1997 Differential evolution vs. functions of the 2nd ICEO *IEEE Conference on Evolutionary Computation* 153–157.
[14] Holland J H 1975 *Adaptation in Natural and Artificial Systems* (The University of Michigan Press)
[15] Zitzler E Deb and K Thiele L 2000 Comparison of multiobjective evolutionary algorithms: Empirical results *Evolutionary Computation* **8** 173–195
[16] Robic T 2005 Performance of DEMO on new test problems: A comparison study *in Proceedings of the Fourteenth International Eletrotechnical and Computer Science Conference* 121–124
[17] Huband S Barone L White L and Hingston P 2005 A scalable multi-objective test problem toolkit *in Evolutionary Multi-Criterion Optimization* 280–295
[18] Tusar T 2008 *DEMO Documentation version 1.2* Jozef Stefan Institute
[19] Poloni C Giurgevich A Onesti L and Pediroda V 2000 Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics *Computer Methods in Applied Mechanics and Engineering* **186** 403–420
[20] Price K Storn R M and Lampinin J A 2005 *Differential Evolution: A Practical Approach to Global Optimization* (Springer-Verlag New York)
[21] CAELinux2008 *http://www.caelinux.com/CMS/*